ARI Research Note 90-107

Causal Models in the Acquisition and Instruction of Programming Skills

Brian J. Reiser
Princeton University

for

Contracting Officer's Representative
George W. Lawton

Basic Research
Michael Kaplan, Director

August 1990

United States Army
Research Institute for the Behavioral and Social Sciences

U.S. ARMY RESEARCH INSTITUTE FOR THE BEHAVIORAL AND SOCIAL SCIENCES

A Field Operating Agency Under the Jurisdiction of the Deputy Chief of Staff for Personnel

EDGAR M. JOHNSON
Technical Director

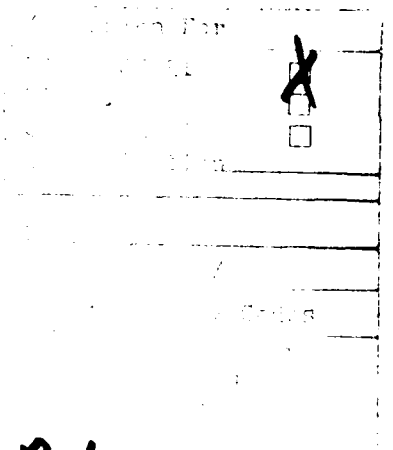
JON W. BLADES
COL, IN
Commanding

Research accomplished under contract for the Department of the Army

Princeton University

Technical review by

George W. Lawton



A-1

NOTICES

DISTRIBUTION: This report has been cleared for release to the Defense Technical Information Center (DTIC) to comply with regulatory requirements. It has been given no primary distribution other than to DTIC and will be available only through DTIC or the National Technical Information Service (NTIS).

FINAL DISPOSITION: This report may be destroyed when it is no longer needed. Please do not return it to the U.S. Army Research Institute for the Behavioral and Social Sciences.

NOTE: The views, opinions, and findings in this report are those of the author(s) and should not be construed as an official Department of the Army position, policy, or decision, unless so designated by other authorized documents.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1a. REPORT SECURITY CLASSIFICATION Unclassified	1b. RESTRICTIVE MARKINGS —
2a. SECURITY CLASSIFICATION AUTHORITY —	3. DISTRIBUTION / AVAILABILITY OF REPORT Approved for public release; distribution is unlimited.
2b. DECLASSIFICATION / DOWNGRADING SCHEDULE —	
4. PERFORMING ORGANIZATION REPORT NUMBER(S) —	5. MONITORING ORGANIZATION REPORT NUMBER(S) ARI Research Note 90–107
6a. NAME OF PERFORMING ORGANIZATION Princeton University 6b. OFFICE SYMBOL (If applicable) —	7a. NAME OF MONITORING ORGANIZATION U.S. Army Research Institute for the Behavioral and Social Sciences
6c. ADDRESS (City, State, and ZIP Code) The Trustees of Princeton University Princeton, NJ 08544	7b. ADDRESS (City, State, and ZIP Code) Office of Basic Research 5001 Eisenhower Ave (PERI-BR) Alexandria, VA 22333-5600
8a. NAME OF FUNDING / SPONSORING ORGANIZATION U.S. Army Research Institute 8b. OFFICE SYMBOL (If applicable) PERI-BR	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER MDA903–87-C-0652
8c. ADDRESS (City, State, and ZIP Code) 5001 Eisenhower Avenue Alexandria, VA 22333–5600	10. SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO.: 61102B PROJECT NO.: 74F TASK NO.: NA WORK UNIT ACCESSION NO.: NA

11. TITLE (Include Security Classification)

Causal Models in the Acquisition and Instruction of Programming Skills

12. PERSONAL AUTHOR(S)
Reiser, Brian J.

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
Interim	FROM 88/09 TO 89/08	1990, August	26

16. SUPPLEMENTARY NOTATION
George W. Lawton, contracting officer's representative

17. COSATI CODES		
FIELD	GROUP	SUB-GROUP

18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
Artificial intelligence; Intelligent tutoring systems; Learning; Mental models; Problem solving; Programming

19. ABSTRACT (Continue on reverse if necessary and identify by block number)
This interim report summarizes the progress we have made on the GIL intelligent tutoring system and the GLEE discovery system for the instruction of programming skills. We describe the current status of these computer systems and the experiments investigating the role of explanations in learning and the use of graphical representations to facilitate problem solving. We also describe the Bat Book problem solving environment for the study of analogies in programming, and our studies of effective human tutors.

20. DISTRIBUTION / AVAILABILITY OF ABSTRACT ☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT. ☐ DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION Unclassified
22a. NAME OF RESPONSIBLE INDIVIDUAL Judith Orsanu	22b. TELEPHONE (Include Area Code) (202) 274–5590 22c. OFFICE SYMBOL PERI-BR

DD FORM 1473, JUN 86 *Previous editions are obsolete.*

Causal Models in the Acquisition and Instruction of Programming Skills

Army Research Institute, Contract MDA 903-87-K-0652 (9/87-8/90)
Behavioral and Social Sciences
Annual Interim Report: Year Two
1 September 1988 to 31 August 1989

Brian J. Reiser
Department of Psychology
Princeton University
Princeton, NJ 08544-1010

1 Overview

This research explores the relation between mental models and rule-based models of problem solving skill. The objective is a theory of the background knowledge that underlies problem solving rules and is needed for explanations. The instructional objective is to investigate how to construct an explanation that incorporates a description of the rule to be learned and its underlying justification.

We are now pursuing a research program that draws on four areas:

GIL-The *Graphical Instruction in LISP* system is an intelligent tutoring system for programming that constructs explanations directly from its problem solving knowledge. We are using GIL to investigate how to represent information for explanations and to conduct empirical studies that examine the effects of the timing and content of explanatory feedback on learning.

GLEE-The *Graphical LISP Exploratory Environment* is a graphical programming environment based on the graphical representations used in GIL, but providing the students more freedom to explore and test their hypotheses. We are using GLEE to investigate how visual representations facilitate problem solving and to examine students' strategies for exploration.

Human Tutors-We are conducting experiments to investigate the tutoring strategies and learning consequences of instruction by human tutors and consultants. Our comparison groups are students learning on their own and in collaborative learning situations.

BATBook-The *Behavioral Analogy Tracing Environment* is an online book and problem solving environment that facilitates students' use of examples in a text book and use of their own solutions to previous problems. We are using BATBook to investigate how students remember previous solutions and adapt them to new problems.

In the second year of this contract, we have made substantial progress in developing the GIL and GLEE systems and have initiated a series of experiments using GIL to study explanations. We have run a large instructional experiment investigating human tutors in a variety of contexts. Finally, we have developed the BATBook system and used it in a first experiment to investigate analogical retrieval.

2 The GIL Tutoring System

We have made substantial progress in the development of GIL this year. We have greatly modified and extended the prototype system that we developed during the first year of the contract. GIL and GLEE together now contain approximately 23000 lines of LISP and LOOPS code. The GIL system contains four major components, the Interface, the Problem Solver, the Model Tracer, and the Explainer. We will describe the advances in each of these components in the following sections. The version of GIL described below is GIL 1.2, dated August 1989.

The personnel on the GIL and GLEE portions of this project during the second year of the contract included the following students and staff (some supported by this contract, others by funding for the Cognitive Science Laboratory): Michael Ranney (Postdoctoral Research Associate), Antonio Romero (research assistant), Alka Tyle (graduate student, Rutgers University), Adnan Hamid (undergraduate), Dan Kimberg (undergraduate), and Marsha Lovett (undergraduate) contributed to various components of the GIL and GLEE code. Shari Landes (research assistant) and John Connelly (undergraduate) ran Experiments 1 and 2, and John Gaskins (graduate student) has an experiment now in progress.

2.1 The GIL Interface

During the second year, we made substantial revisions in the prototype GIL interface developed during the first year. These revisions were suggested by several series of pilot subjects and feedback from our colleagues at a workshop of ITS and CAI researchers in March 1988 (sponsored by the Cognitive Studies for Educational Practice Program of the McDonnell Foundation). These changes concerned a facility for typing new data directly into the graph instead of separate input windows, the

use of a "click and drag" method for bringing functions and nodes into the program graph (the method used for moving icons in the Macintosh Finder and therefore instantly recognized by most novices), the placement of a problem statement window, explanation windows, buttons, and many other details. The resulting interface is now quite easy for novices to master. We find that students in our experiment who have no prior programming experience can learn to use GIL with a short five minute demonstration and can work through two chapters of material without having any questions on the interface.

2.2 Production Rule Problem Solver

The architecture of the production rule problem solver was developed in the first year. The major advance in the production rule system in our second year has been the extension of the production rules and plans to complete the first two chapters of problems. GIL 1.2 contains 182 rules (approximately 7500 lines of code) and 13 plans. The rule set contains 78 forward enable rules, 84 backward rules, and 20 achieve rules. The GIL problem solver differs from other production rule problem solvers used in intelligent tutoring systems in a number of respects. First, the problem solver combines reasoning in two directions. The system contains rules for reasoning forward from the given data toward the goal, and rules for doing goal decomposition by reasoning backward from the goal toward the given data. Another unique characteristic is the use of inheritance hierarchies in the semantic predicates in the pattern matcher and in the organizations of the rules themselves. If more specific information has been inferred in a particular problem, the more general predicate in a rule will match against the subordinate predicate in a proposition in working memory. In this way, the propositions included as the conditions of rules can be written at the level of abstraction appropriate for the type of rule. The hierarchy of rules makes it possible to organize similar problem solving steps into families of related rules, and to encode more specific versions of general strategies. The problem solver traverses the hierarchy to find the most specific rules that match the current facts in working memory.

The problem solving plans in GIL represent important sequences of steps to achieve a particular type of subgoal. Many of GIL's rules either initiate a plan or are applied only if a particular plan has been undertaken. Thus, GIL can use these plans to explain the larger context in which a step occurs.

The first two chapters in the curriculum contain 15 problems currently being used. Now that the rules are stable, more problems can be developed simply by adding appropriate working memory descriptions. These problems contain up to 14 unique solution graphs each, and up to 1720 different paths or different sequences

of steps that a student can take in each problem. We have developed the Pathfinder Solution Generator (described in the next section) to test the solutions permitted by GIL for these problems and ensure that students can solve each problem using any possible sequence of steps.

2.3 The "Pathfinder" Solution Generator

The flexibility allowed by GIL posed serious difficulties for debugging the problems in the curriculum. Even problems that contain only 5 or 6 functions in the solution may be solved with as many as 14 different solution graphs. Since GIL allows both forward and backward reasoning, and allows the student to pursue any path at each point in the problem, there are typically 4-8 moves possible at any point. In some of the more complex problems, there may be more than 1700 different sequences of correct steps logically possible. Thus, it is simply not feasible to "manually" test all possible sequences to ensure that they are allowed by GIL.

For this reason, we developed a solution generator program, called "Pathfinder", to help automate the debugging process. The Pathfinder exercises the problem solver through each problem to construct every solution allowed by the rules and thereby determines all sequences of steps that are permitted. The Pathfinder finds any dead ends (sequences of steps that result in a state from which it is not possible to reach a solution), missing paths (a sequence of correct steps not allowed by the rules), extra steps (rules which match when the solution is complete), and rule redundancies (a case in which two rules match to a state but the two rules have the same effect and therefore are redundant). The Pathfinder program made it possible to debug and fine-tune our rule set during this year and finalize the curriculum for the first two chapters of material. Our Pathfinder results guarantee that students can solve each problem using any order of steps.

2.4 The Compiled Problem Spaces Model Tracer

We implemented a major improvement in the speed of GIL's response with the use of compiled problem spaces. Since the Pathfinder has already worked out all possible sequences of production rule firings that can be used to solve the problem, the GIL Model Tracer can use this generated problem space to trace the student's reasoning. Thus, rather than matching production rules on each cycle, the Model Tracer simply accesses the stored conflict set of matched productions and examines that to evaluate the student's step. This technique results in extremely fast responses to each student action, less than 2 sec/step.

2.5 The Conditionals Curriculum: A Plan-Based Model Tracer

A major focus of our research efforts in the second half of this year has been on the representation of conditional expressions (Chapter 3 in our curriculum). Our experience in building the production rule problem solver for the first two chapters of material has revealed some limitations in the production rule approach when students are given complete freedom to work on any part of the problem at any point. The large number of contexts in which a particular step may be taken results in a large increase in the number of rules that are necessary. Our attempts to build a prototype rule set for conditional expressions suggested that a greater reliance on problem solving plans would facilitate the construction of the problem solver, and would help us to tackle the issue of providing more global explanations.

We have constructed an extension of the production rule problem solver that uses plans rather than individual production rules. The problem solver constructs a complete *and/or* tree of steps for a problem. We completed work on the plan-based problem solver during the second year, and are currently writing the plan set necessary for the Chapter 3 problems.

We have also constructed a prototype model tracer that uses the plan representations. In the next quarter we will fine-tune the interface and model tracer to work with these new representations.

2.6 The GIL Explainer

GIL has two features which make it unique as a tutoring system. The first is the graphical representations that students use to build programs. The second is the GIL explainer that dynamically generates feedback directly from its problem solving knowledge, rather than relying on canned text written by the system's programmers. Furthermore, the GIL explainer does not require a bug catalogue such as found in most tutoring systems. GIL responds to student errors and requests for guidance by finding the relevant problem solving rules and plans and generating explanatory feedback to guide the student's reasoning. If an error in a student step is found, GIL's explainer analyzes the discrepancies between the student's step and the closest matching correct rule and offers suggestions to the student about how to improve the step. Explanations may draw upon the problem solving rule, general knowledge about the operator being used, and the higher-level plan of which the step is a part.

We constructed the prototype explainer system in the first year of the contract. In the second year, we redesigned and extended the explainer into a robust system. The explainer contains 15 general bug categories that diagnose whether the error

is a legal error (in which the step in the program does not correctly manipulate the selected data) or a strategic error (in which the chosen step is a legal LISP operation but is not strategically useful). In constructing the explanation GIL may access general information about the LISP operator, the particular rule specifying the strategic constraints for applying that operator and its effects on the current problem state, and the inferred student plan. GIL first describes what is good about the student's step and then points out the ways in which the step is in error or could be improved by explaining how the student's input, output, or function deviates from the properties in the correct rule. An example of GIL's response to a strategic error is shown in Figure 1.

The explainer contains a natural language generator for propositions which enables it to express any proposition from the condition or action component of a rule as an English clause. The explainer is sensitive to focus, so it will express a particular proposition differently depending on which argument of the proposition is the focus in the current explanation context. The explainer can also handle embedded relations, in which it explains the connection between two objects in the problem that are connected by a chain of propositions (e.g., "the first element of the reverse of the original input list"). The explainer also contains facilities for recognizing and pruning redundant information. For example, if it decides to point out that a particular student input should be a list, it is smart enough to exclude from its explanation other violated properties that cannot be satisfied if this more basic property is violated.

2.7 The GIL Graph Viewer

In examining the data from subjects in our experiments (described below), we found it important to observe the appearance of the graph, rather than simply relying on a list of its contents. Often the layout of the graph is a clue to students' intentions. For this reason, GIL now stores the state of the program graph after each correct or incorrect step. The Graph Viewer program can be used to replay a subject's series of steps for a problem. An example is shown in Figure 2.

2.8 Use of GIL within the ARI Basic Research Office

During the second year of the contract, we began a collaboration with Susan Viscuso, a National Research Council Associate at the Army Research Institute. Dr. Viscuso is interested in investigating the manner in which graphical representations facilitate problem solving and the types of difficulties students have learning to plan LISP solutions and learning the syntax of LISP. She is comparing one group of subjects

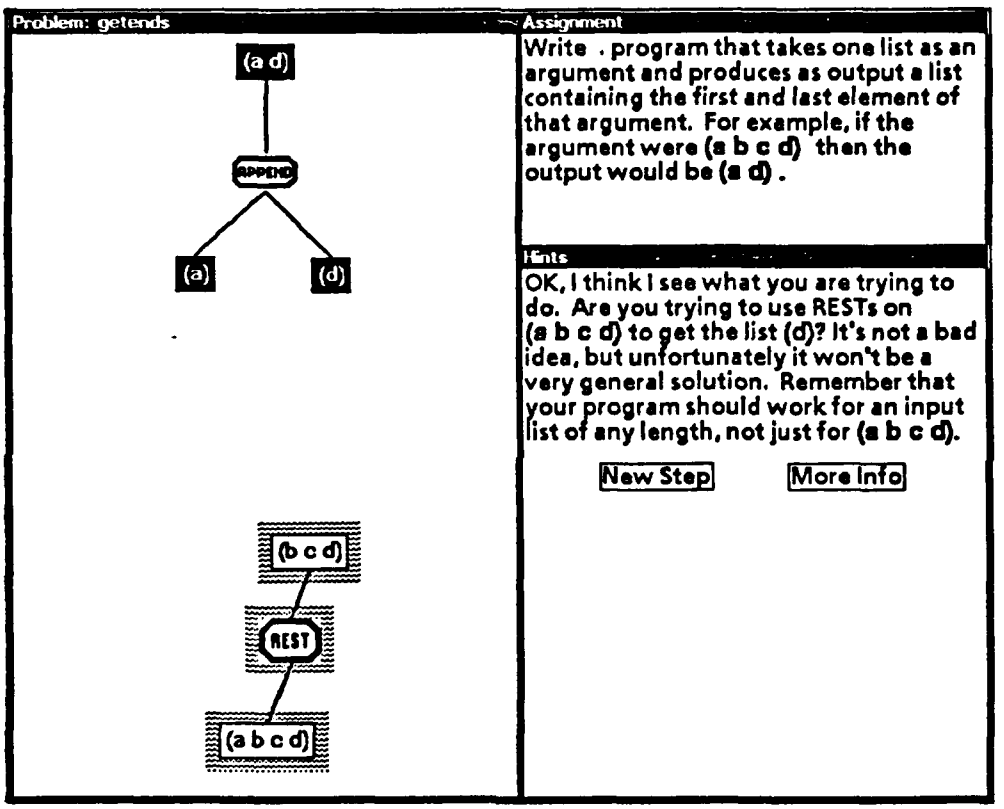


Figure 1. An explanation constructed by GIL in response to a strategic error.

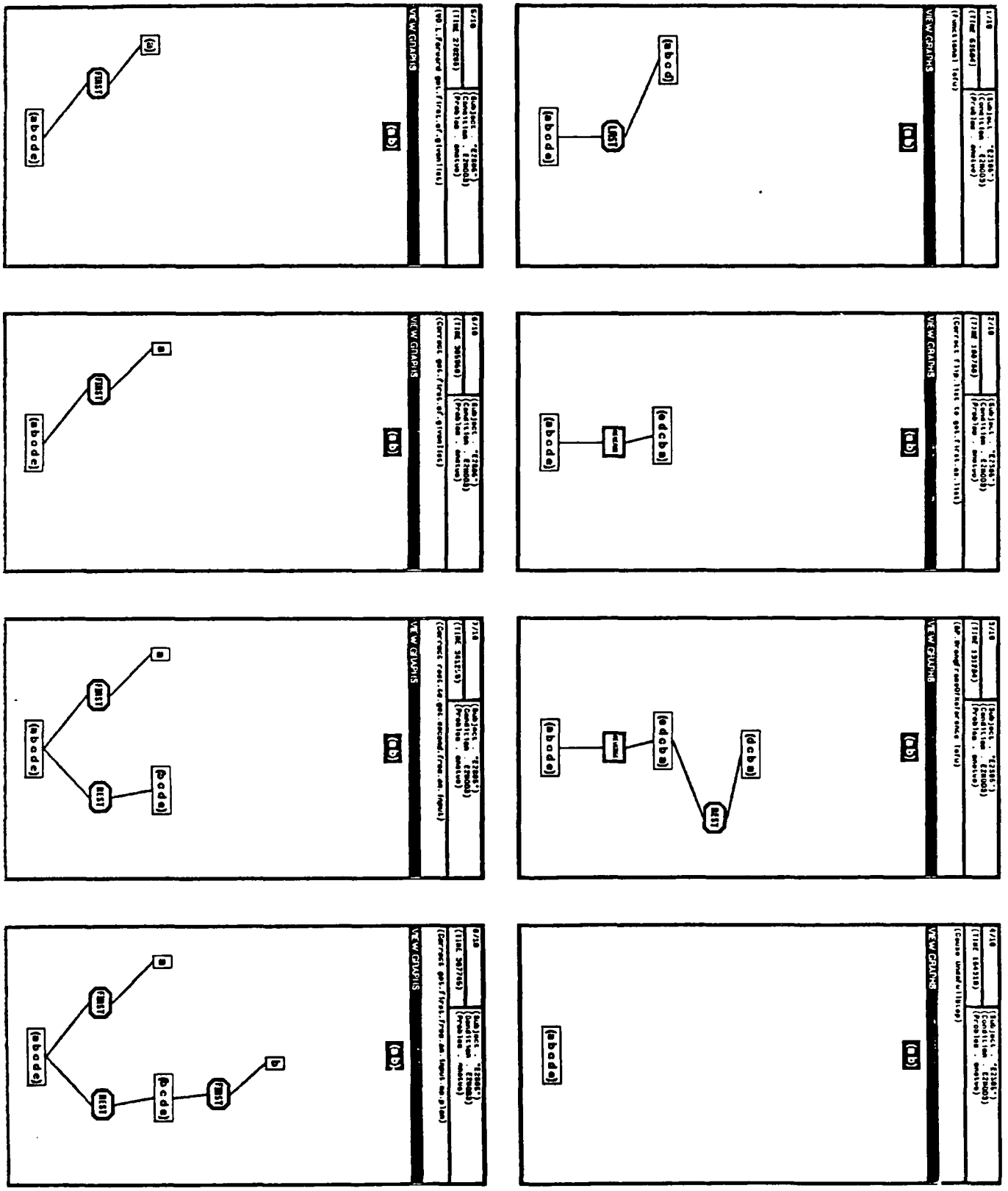


Figure 2. A GIL Graph Viewer display of a portion of a student's solution to the onetwo problem.

learning to program in GIL with another group of subjects who program in standard LISP but are given an instructional text that demonstrates LISP functions using GIL graphical representations in addition to standard text LISP. In her experiment, GIL subjects are also trained in standard LISP by translating each GIL graph into text LISP. This provides a common language in which to evaluate the learning of both groups of subjects.

3 The GLEE Discovery System

The GLEE system is an exploratory system based on the graphical representations used in GIL. We began construction of GLEE during the second year of the contract. A complete version is now ready and we are about to test GLEE with subjects. GLEE provides students the graphical representations of GIL, but without the tutoring assistance and the model tracing constraint. This enables students to explore.

There is much rhetoric in the education field about the virtues of discovery and exploration. However, there has been little attempt to demonstrate the important features of a successful exploratory system. Furthermore, there has been little empirical demonstration of the advantages of learning by discovery. GLEE and GIL provide us the opportunity to investigate these issues. There is no clear guidance from the education community on the design of an effective exploratory system. There is much more to a successful exploratory system than simply removing the tutorial guidance from a tutoring system. We are attempting to develop a set of pedagogical principles for an exploratory system in our work on GLEE.

The GLEE system provides students more freedom than students in GIL. First, students in GIL are constrained to include three components in each step — a function, the input, and the output. In GLEE, the student is not constrained to take steps in this fashion. Instead, the students can bring in as many functions and data nodes and in whatever order that is desired. Because of this, GLEE can not infer whether a data node is intended as input or output or will be attached to a function that has not yet been brought into the graph. Therefore, unlike GIL, GLEE does not draw any links between functions and data nodes. Instead the student links any nodes desired by clicking on a node and dragging a line over to another node and releasing the mouse button. Furthermore, unlike GIL, the student is not constrained to build graphs only from the bottom up and top down. Instead, the student can place any functions and data nodes anywhere in the graph and link them in whatever order desired. Figure 3 displays a partial solution in GLEE. Note that the student has begun two forward paths, and has also constructed a portion

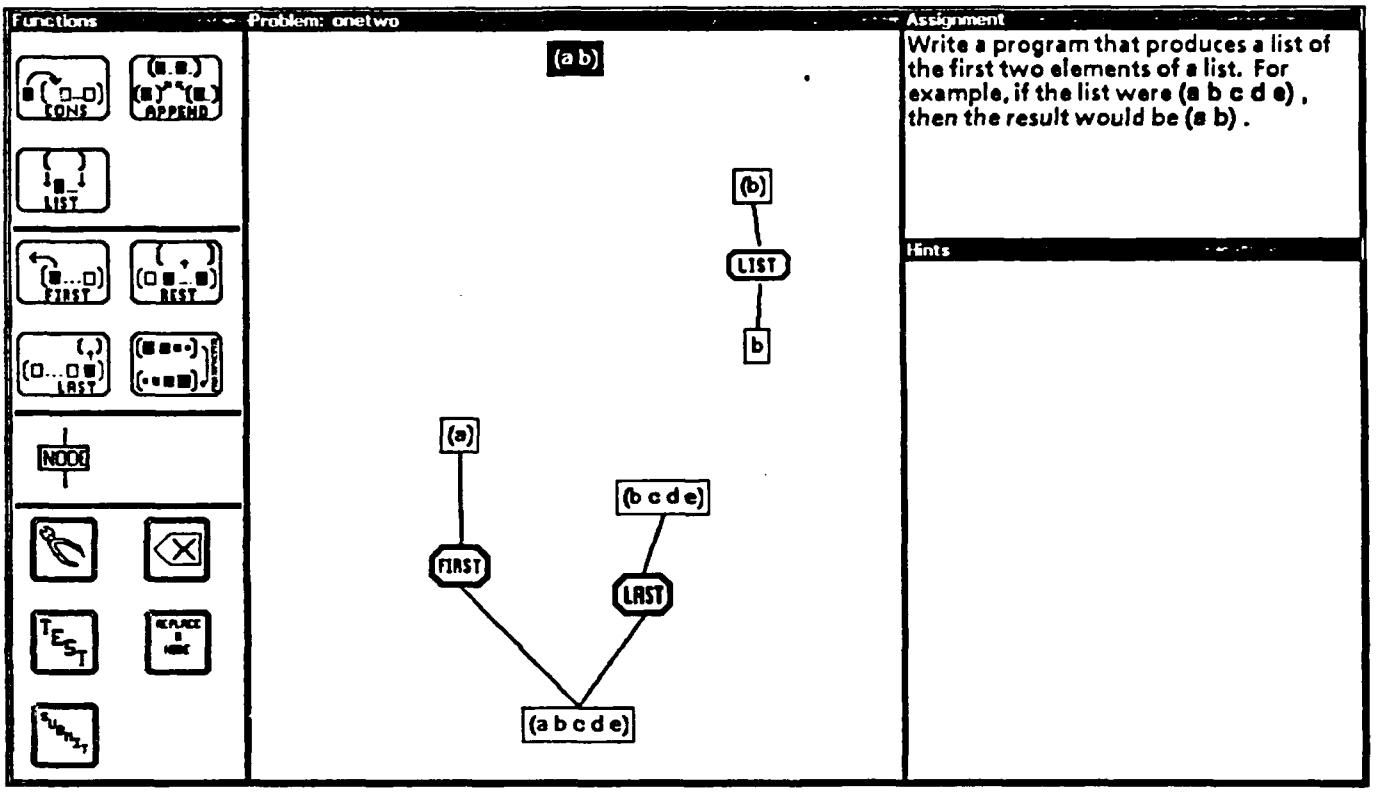


Figure 3. A partial solution in the GLEE exploratory environment.

of the graph that has not yet been connected (the LIST of b to get (b)).

In GLEE, students are given the opportunity to construct graphs in arbitrary order and without constraining them to correct errors as they are made. For this reason, it is important to provide students with methods for deleting and editing portions of a graph. In GIL, since errors are corrected as soon as they are made, the only delete capability is the "Oops" button which can be used to delete the last portion of the current step, or to delete the entire step. GLEE contains three types of graph modification routines. First, the delete key (shaped like a typewriter backspace key) can be used to delete any node in the graph. The student selects the delete key and then clicks on any function or data node in the graph. That node is then deleted, and any links to the node are removed. Second, GLEE contains a "Replace" key. After selecting a Replace, the student is prompted to click on an item in the graph. If the student clicks on a function, then a click on a function in the Function Menu causes the selected function to replace the function in the graph. If the student clicks on a data node, then the node changes to an empty data node, and the student can simply type in the new value. The third type of deletion is the "Break Link" key (the wire cutters icon in the menu). This is used to break a link between two nodes in the graph.

We argue that an effective discovery system must do more than refrain from telling students when they make mistakes. Unless the system provides the student the opportunity to test hypotheses, there seems little chance of learning from the exploration. Thus, a crucial component of discovery systems is the ability to elicit feedback from the system when desired, so that the student can test hypotheses. For this reason, GLEE contains a "Test" button. When selected, the student can then click on any node in the graph. The system then begins with the initial input and tests the paths up to the target node. If GLEE finds any data nodes that are incorrect, or any functions that cannot be applied to the input data, then it points out the error. Figure 4 displays GLEE's response to an error. Of course, GLEE responds to legal errors, and does not comment on the student's strategy.

GLEE also contains a "Submit" button that is used to submit a completed solution. When the student has tested a solution and feels it is correct, the Submit button will test the student's program against a set of examples associated with the problem to determine whether the program computes the correct result for each example. If the student's program satisfies the problem constraints, then GLEE presents the next problem in the curriculum. If the student's program only works for the original example but is not general enough to work for the other examples, GLEE informs the student of the example that does not work and gives the student the opportunity to attempt to modify the program.

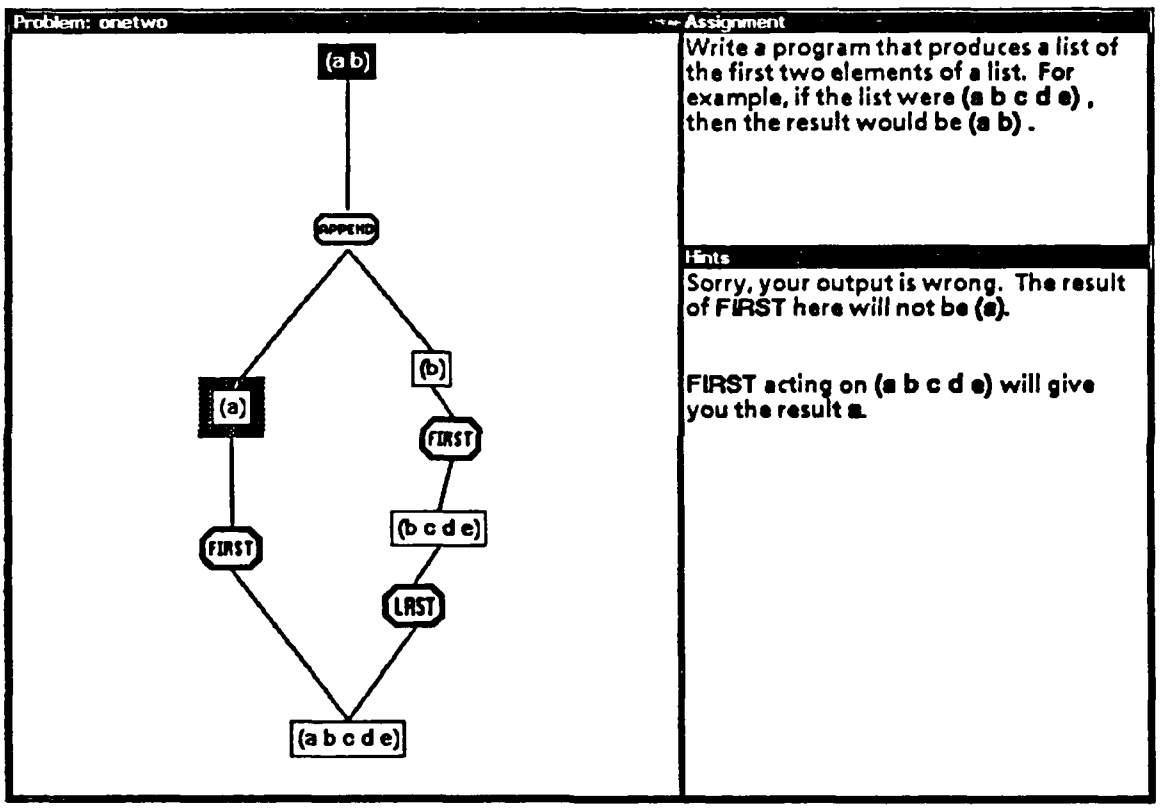


Figure 4. Feedback on an incorrect solution in GLEE.

4 Experiments Using GIL and GLEE

4.1 Experiment 1: Comparison of GIL and Standard LISP

We have begun to evaluate the effectiveness of GIL. Our first study, presented at the 1989 AI and Education conference, compared students learning to program with GIL with another group of students learning the same programming concepts in traditional text-based format. Students found the program graphs much easier to construct than standard LISP syntax. However, even after the control group students had mastered the syntax of LISP definitions, there was a substantial advantage for using GIL to solve problems. Students working with GIL solved comparable problems in approximately 1/4 the time of students working in standard LISP. We believe that the faster learning of the GIL students is due to two aspects of the environment: the model tracing nature of GIL and the nature of interacting with GIL's graphical programming environment.

The model tracing nature of GIL provides feedback for students somewhat similar to the type of feedback that human tutors provide. GIL's feedback achieves the same purpose as a human tutor's constant feedback: it keeps the student from going too far off the track and minimizes the consequences of errors. The GIL feedback points out whenever a step is in error. This feedback helps the student locate the error by marking the portion of the student's step that needs to be modified in the graph. In addition, the feedback briefly describes how the step can be improved. The feedback is in the form of a hint, in that it describes the nature of the difference between what the student has and what is needed, but does not provide the correct answer (unless the student requests "More Info"). In addition, GIL's feedback distinguishes between a variety of errors in reasoning about LISP's behavior (legal errors) and poor strategies (strategic errors). The error feedback and hints appear to greatly reduce the students' time to construct a solution.

Another advantage of the GIL tutor that appears to account for its effectiveness is that the environment is more congruent with the reasoning students need to do to construct programs than the standard LISP environment is. Students can more easily understand how LISP works and more easily plan algorithms using GIL. The most striking evidence for this concerns their use of forward and backward reasoning — subjects showed a strong tendency to work forward from the given data toward the goal. In contrast, the order of the functions in text form of the code corresponds to a complete top-down or backward solution. If subjects' reliance on forward steps in GIL is a true representation of their reasoning, then novices do not appear to plan their programs in the order in which the functions appear in the completed solution. Forcing students to enter their code in the outside-in left-to-right fashion required

by standard LISP interpreters forces students into a different strategy than the one in which they can work through the algorithm in the order in which it transforms the data.

4.2 Experiment 2: Content of Explanations

A second GIL study, conducted by John Connelly, demonstrates the importance of GIL's explanations. This experiment employed two modified feedback conditions, in addition to the standard explanatory GIL condition. In the minimal feedback case, GIL merely indicated whether each step was correct or not, and provided students the opportunity to request the correct step, without an explanation of the error or the aptness of the correct step. In the location feedback condition, GIL indicated which part of the step was incorrect and indicated how to fix it upon request, but again provided no explanations. Subjects in these conditions relied more on the second level of help to be told how to fix the error. In spite of this greater reliance on being told the right answer, these subjects exhibited longer error-fixing episodes and poorer performance on post-tests than those receiving explanations upon errors. Location feedback subjects performed better than minimal feedback subjects, but both groups fared worse on all measures than the explanatory GIL feedback subjects. The study suggests that GIL's facilitation of students' learning is not merely providing feedback on the correctness of the steps or in leading students to a solution by telling them the answer. Rather the positive effects of GIL lie in the way in which the explanations enable students to fix their errors and understand why their fixes are successful.

4.3 Experiments Now in Progress

There are a number of studies now in progress in our laboratory. In one study, being conducted by John Gaskins, we are investigating the transfer from the graphical representations used in GIL to the standard text-based syntax of LISP. Students solve each problem in GIL and then translate the program graph into text LISP. A control group solves each problem in standard text LISP. Even though the control group has been using text LISP to solve all their problems and therefore will have more experience with this form of LISP, we predict that the GIL group will perform better on post-tests in text LISP, because they will have acquired a better understanding of how functions operate in LISP and will be better able to construct algorithms to solve problems.

A second experiment continues our investigations of the effectiveness of explanations in learning. The procedure is similar to Experiment 2 described above with

an important modification. In Experiment 2, subjects were given the "More Info" option in the Minimal Explanation and Location conditions. In these conditions, selecting this option simply told the subject how to fix the step, without explaining why the fix was necessary or effective. We found that subjects were more likely to rely on the More Info option in these conditions than in the standard GIL condition. One possible explanation for the superiority of the GIL condition is that when subjects are given hints they try to fix the error themselves, while in the Minimal Explanation and Location conditions subjects were less motivated to try to reason through the fix of the error and therefore relied more on being told what to do. It may be that the effects of the experiment were due not to the explanations per se, but rather to the subjects relying on being told what to do instead of discovering how to fix the errors themselves. In the current experiment, subjects are never given the More Info option, and therefore must fix all of their errors themselves. A difference in performance between the groups in this experiment will be stronger evidence for the role the explanations themselves are playing in the subjects' acquisition of the target knowledge.

In the next quarter, we hope to begin pilot testing of GLEE with subjects and comparisons of several versions of GLEE. These experiments will investigate the effectiveness of intermediate products in the graphical representations and the utility of stepwise feedback.

5 Experiments With Human Tutors

Another component of our research program is the investigation of effective human tutors. We are currently studying human tutors in a variety of instructional contexts in order to investigate tutoring strategies, the timing of feedback, contingency of feedback on models of the individual student, and the type of explanations used by tutors.

In one study, currently in progress, we examined students learning to program in a variety of learning situations. All subjects read the first three chapters of our LISP programming textbook (*Essential LISP*), which covered the topics of using LISP functions to manipulate numbers and lists, defining new functions, and writing programs using predicates, conditionals, and logical functions. Subjects read the text and solved the problems in each chapter, using a LISP interpreter and a simplified screen editor to modify their programs. The experiment required between 3 and 5 sessions totalling between 6 and 15 hours.

The study included four learning conditions: No Tutor, Tutor, Consultant, and Collaborative Learning. The No Tutor subjects were instructed to read the text

and solve the problems on their own, and ask questions only if necessary. At the end of each chapter, these subjects presented their solutions to the experimenter, who graded each solution as correct or incorrect, whereupon subjects were given an opportunity to attempt to correct their mistakes. In the Collaborative Learning condition students worked in pairs with the same instructions as the No Tutor subjects. The Tutor subjects worked through the same material with an experienced human tutor. In the Consultant condition, the tutors were present only a few feet away in an adjacent room in the laboratory suite, within sight of the subjects. Subjects in the Consultant condition were instructed to ask as many questions as they wished of the human tutors. When asked, the tutor would sit down next to the subject and help with whatever problem or confusion the student had encountered, until the tutor felt the topic was completed, typically 5-10 minutes later. The purpose of this condition was to compare a natural tutoring situation with one in which the tutoring interactions were initiated only by the student. We employed two tutors, one male and one female, both Princeton undergraduates. Subjects were matched with the tutor of the same gender in the Tutor and Consultant conditions.

The results demonstrated a dramatic advantage for the Tutor subjects. The subjects completed the material approximately 40% faster than the No Tutor subjects, and completed the problems with only one-third the number of solution attempts. The Collaborative Learning subjects were only slightly faster than the No Tutor subjects. Learning times for the Consultant subjects fell between these groups, slower than the Tutor subjects but faster than the No Tutor and Collaborative subjects. Interestingly, although the subjects in this condition asked relatively few questions, (approximately 3 questions per chapter, or 1 question every 45 minutes), this help at presumably crucial points in the subject's problem solving resulted in faster learning than the No Tutor subjects.

We are currently examining the tutoring protocols to analyze the intervention and tutoring strategies employed by the tutors. Our initial analyses (presented at the 1989 AERA conference) suggest a view of tutors as safety nets for learning by doing. Although the solutions to these programming problems were typically programs no longer than 5-10 lines, many were relatively difficult and could require up to 45 minutes to solve. The sessions were highly interactive, as tutors helped the students set goals, pointed out errors, provided guidance in fixing the errors, and provided hints for steps in the solution. The students relied on continual feedback from the tutors. Tutors reacted to each step with confirmations, questions, prodding, asking for justifications, etc. In most cases, the tutor's feedback, although indirect, enabled subjects to quickly determine whether a solution path was correct or likely to succeed. The tutors, through questions and hints, were able to focus students on the part of a solution that needed elaboration or repair.

The tutors' feedback varied in its timing and content. In some cases, the tutors just prompted the student to rethink a step, whereupon the student would initiate a repair of the partial solution. In other cases, the tutors interrupted to tell the students something that was missing or incorrect. In contrast, the tutors also let some types of errors go, returning to them at the end of the solution when the student was ready to check the program by running it. Thus, the tutors appeared to modulate their responses depending upon the potential learning consequences of the error. Tutors quickly corrected errors that would be distracting and might lead to floundering, and did not comment during the solution on errors that might lead to productive learning episodes later. Instead, they made sure that the subject discovered the error at an appropriate point, and then helped to diagnose the problem and fix the solution.

Through hints, leading questions, and continual confirmatory feedback, the tutors guided the students' problem solving, and prevented the students from reaching error states from which it would be too difficult to recover. This enabled the students to master the material more quickly than students working on their own. Our consideration of these results suggests similar pedagogical benefits for the model tracing design of intelligent tutors, as well as ways in which human tutors have not yet been paralleled in intelligent tutors (Reiser, 1989c).

6 BATBook: An Environment to Study Analogies in Programming

An important issue that has arisen in constructing explanations is the use of specific examples and general principles in tutorial explanations. There are few guidelines to determine the situations in which a tutor should provide an explanation based on general principles, and when reminding the student of a previous solution would be advantageous. Furthermore, the circumstances governing the retrieval of previous examples from memory is a central controversy in current research on analogy. We have developed BATBook (Behavioral Analogy Tracing Environment), an online book and problem solving environment, to examine how subjects use examples in text and memories for their previous solutions during learning.

We designed and constructed our original version of BATBook during the first year of the contract. During the first year, we ran an experiment using this version of BATBook to teach two chapters of material. During the second year, we completed this experiment by extending BATBook to cover three chapters of material so that we could investigate students' use of examples on a more difficult selection of material. Much of our work on this part of the project has been devoted to analyses

of the results and the design of a follow-up experiment (described below). During the second year, we also extended the design of BATBook to include facilities for searching text examples.

The BATBook environment runs on Sun workstations in the SunView window system. BATBook is written in C, Franz LISP, GNU Emacs LISP, and the Lex lexical analyzer. The majority of the interface for displaying and searching text is written in C, using Lex string handling functions. The problem solving environment for LISP is written in Franz LISP, including the modified LISP interpreter and solution analysis routines. The current version of BATBook contains a total of 5900 lines of C code, 3200 lines of Franz Lisp and GNU Emacs Lisp code, and 90 lines of Lex code. BATBook was written by Jerry Faries (graduate student), with assistance from Eric Ho (research assistant) and Antonio Romero (research assistant).

The basic paradigm in BATBook is that the subject reads a textbook on the computer screen, interspersed with example solutions to problems and with problems for the subject to solve. Subjects progress forward and backward through the text by pages and can search the expository text and worked solutions for any particular content they can specify. While working on exercises, subjects are free to search the instructional material in the text, the examples contained in the text, and their past work. Subjects store their completed solutions and can request to see a previous solution at any time. They are free to retrieve as few or as many examples as they find useful. BATBook records all interactions, including the time spent reading each page, searches of the text, all problem solving work, and successful and failed searches of text examples and previous solutions.

In this way, we can examine the reminding and use of previous examples in a situation in which subjects are engaged in learning a new domain and solving problems. Retrieving a study example, an example of an error encountered previously, or a successful solution are all methods in the student's battery of learning strategies. With this method, we can investigate the use of examples in a natural context, where the examples are used as part of the problem solving. Furthermore, the similarities subjects notice between problems are made explicit by the descriptions they use to request the retrieval of a previous example.

We have now run an experiment using the BATBook electronic book environment examining the circumstances under which subjects decide to access a previous solution as an analogy to help solve a new problem. In our first experiment, novices used BATBook to read the first two chapters of our LISP textbook, *Essential LISP*. Each chapter of the text contains several short sections of instruction followed by problems that apply the knowledge introduced in the section. In addition to the regular problems in the text, we added a second set of problems to Chapter Two. These problems were designed so that each problem contained a cover story simi-

lar to a previous problem and was structurally similar to a different problem from the first half. We were interested in whether subjects would retrieve solutions from the first half of the chapter while working on the second half, and whether these retrievals would be governed by surface or structural similarities. In addition, we included two variations on the method for storing previous solutions. One group provided a verbal label for each solution they constructed and later could use this verbal label to retrieve a previous solution, while the other group did not label their solutions and could retrieve them only by referring to the problem description or content of the solution. Figure 5 displays a student retrieving a previous solution using a word from the problem statement as a search key.

The results (presented at the 1988 Cognitive Science Conference) provide strong evidence that people in a clearly defined problem solving situation are sensitive to and are able to make use of structural correspondences between problems. The subjects were rarely misled by the superficial correspondences and were able to identify and make use of structural similarities between problems. They may not have had all the well formulated rules they needed, but they were sensitive to the structural nature of the problems and could therefore detect functional similarities. The efficient use of examples requires subjects to have encoded the relevance of particular examples, and to remember enough about them to generate successful retrieval descriptions.

During the second year of the contract, we have extended this study by running another group of subjects through the first two chapters as well as a third chapter. The third chapter contained 13 questions with an additional 12 test questions. Each test question had one structural antecedent and one superficial antecedent. This additional data was collected for two reasons. First we wanted to better assess differences between the label and non-label group. In addition we wanted to examine whether the trends would hold with more complex problems involving more sophisticated programming structures.

The addition of this new data gives us confidence in the main finding that subjects are able to identify structurally related previous problems despite superficial differences between them. This finding held even for the more complicated problems. The difference between the labeling conditions, however, disappeared.

During the second year, we also modified the BATBook environment to conduct a second experiment which will compare the memory for previous problems depending on whether one actually *solves* the earlier problems or only *studies* worked-out solutions. In our first study we found evidence that subjects solving problems are able to notice significant similarities. We have suggested that novices may have a memory for the previous problems based on the kinds of problem solving obstacles, discriminations, and decisions made while solving the problem. If this is true

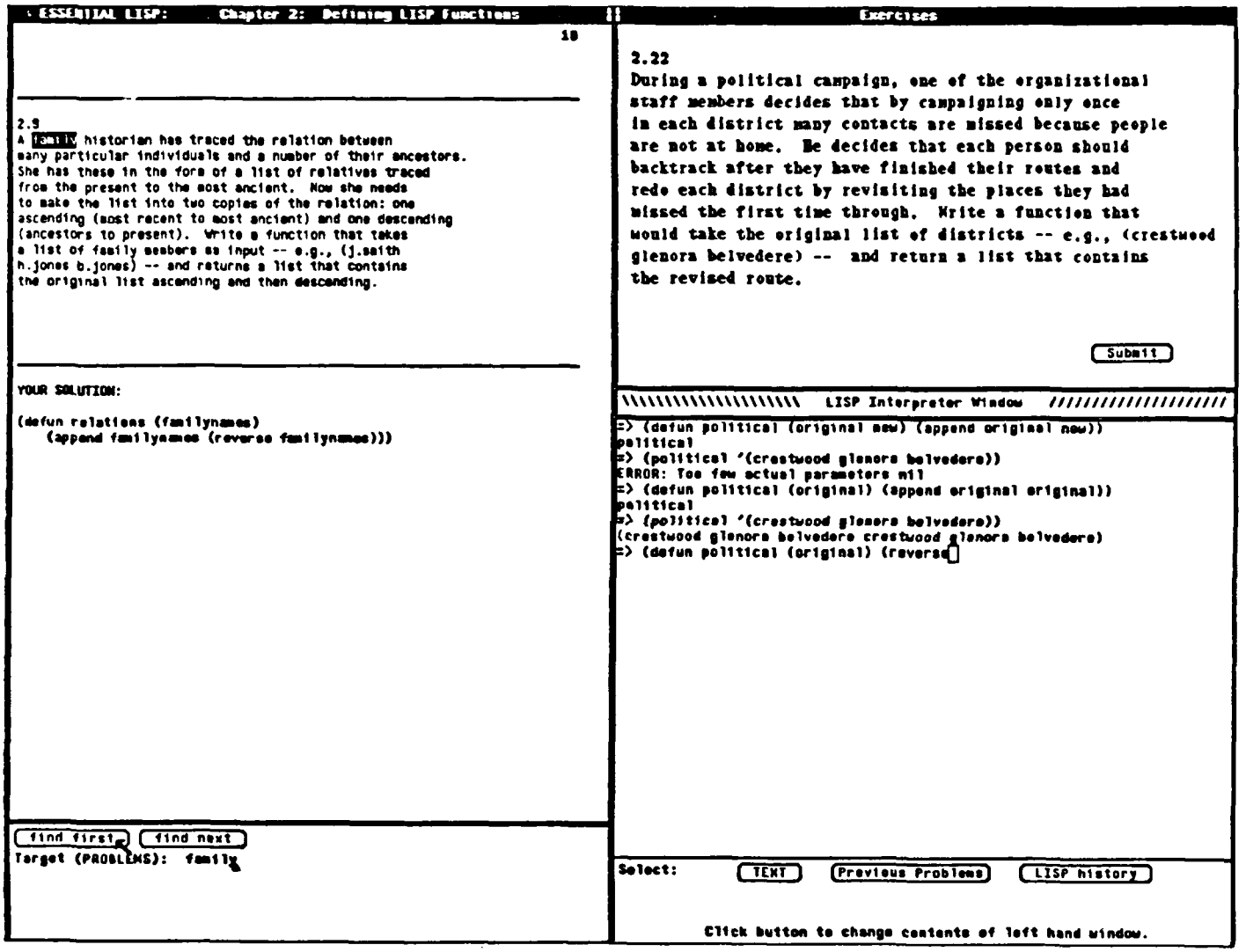


Figure 5. Retrieval of a student's earlier solution in BATBook.

then we should expect a significantly greater amount of structurally related search activity for subjects who solve a problem that for those who simply study it. Another motivation for this study is that earlier work by Ross demonstrating a greater influence of superficial similarities. His subjects were only asked to study the antecedent problems and solutions. We consider this to be a crucial difference between our study and his and speculate that it may explain the discrepancies between our respective findings. We plan to begin this experiment in the next quarter.

7 Publications and Presentations of the Research

7.1 Invited Papers and Colloquia

The Principal Investigator was an invited participant in several workshops on cognitive science and instruction during the second year of the contract. Reiser was an invited participant in the NATO Science Committee Workshop on New Directions in Educational Technology, November 1988, to discuss the future of research on educational technology. Reiser also presented an invited paper at the Seminar on Computers and Learning sponsored by the Social Science Research Council, held June 1989 in Tortola, BVI.

Colloquia describing this research were presented by the Principal Investigator at the University of Chicago, McGill University, BBN Laboratories, and Bell Communications Research.

7.2 Conference Presentations and Publications

We presented our research in a number of international education, AI, and cognitive science conferences during the second year of this contract. Talks were presented at the American Educational Research Association, the National Educational Computing Conference, the Cognitive Science Conference, and the Artificial Intelligence and Education Conference.

Faries, J. M., & Reiser, B. J. (1988). Access and use of previous solutions in a problem solving situation. *Proceedings of the Tenth Annual Conference of the Cognitive Science Society*, Montreal, pp. 433-439.

Reiser, B. J. (1989a). *Pedagogical strategies for human and computer tutoring.* Paper presented at the Annual Meeting of the American Educational Research Association, San Francisco, CA, March 1989.

Reiser, B. J. (1989b). *An intelligent tutoring system for facilitating students' reasoning about computer programs.* Paper presented at NECC '89: National Educational Computing Conference, Boston, MA, June 1989.

Reiser, B. J. (1989c). *Pedagogical strategies for human and computer tutoring.* Technical Report #38, Cognitive Science Laboratory, Princeton University.

Reiser, B. J., Ranney, M., Lovett, M. C., & Kimberg, D. Y. (1989). Facilitating students' reasoning with causal explanations and visual representations. In D. Bierman, J. Breuker, & J. Sandberg, Eds. *Proceedings of the Fourth International Conference on Artificial Intelligence and Education.* Springfield, VA: IOS.

Faries, J. M., & Reiser, B. J. (in press). *BATBook: An online book and problem solving environment for the study of skill acquisition.* Technical Report, Cognitive Science Laboratory, Princeton University.

Ranney, M., & Reiser, B. J. (in press). Reasoning and explanation in an intelligent tutoring system for programming. To appear in the *Proceedings of the Third International Conference on Human-Computer Interaction,* Elsevier Science Publishers, 1989.

Reiser, B. J. (in press). Problem solving and explanation in intelligent tutoring systems: Issues for future research. To appear in E. Scanlon & T. O'Shea, Eds., *Proceedings of NATO Workshop on Educational Technology,* Springer-Verlag.

Reiser, B. J., Kimberg, D. Y., Lovett, M. C., & Ranney, M. (in press). Knowledge representation and explanation in GIL, an intelligent tutor for programming. In J. Larkin, R. Chabay, & C. Scheftic (Eds.), *Computer Assisted Instruction and Intelligent Tutoring Systems: Establishing Communication and Collaboration,* Hillsdale, NJ: Erlbaum.

7.3 Awards and Offices

The Principal Investigator was awarded the Princeton University Class of 1936 Bicentennial Preceptorship, 7/1/88-6/30/91. This award provides the PI with one year's sabbatical leave, to be taken from September 1989 through June 1990. The PI will spend the year in residence at Princeton University to pursue research on the contract while released from teaching and administrative duties.

Reiser was also elected President of the Special Interest Group on Artificial Intelligence and Education of the American Educational Research Association at the 1989 Annual Meeting.